Amendments to the Claims:

1. (Currently Amended) A method of profiling code for an execution environment in which latency exists between an execution event and detection thereof, the method comprising:

identifying an ambiguity creating location in the code, the ambiguity creating location being an operation reachable from a plurality of program execution paths;

executing the code on a processor;

detecting the execution event; [[and]]

backtracking a displacement ~~within an unambiguous skid region~~ from a detection point in the code coinciding with the detection of the execution event to a preceding operation;

determining that the ambiguity creating location is not encountered while backtracking; and[[,]]

wherein the displacement is based, at least in part, on a type of operation appropriate to have triggered the execution event.

2. (Cancelled)

3. (Currently Amended) The method of claim 1 further comprising,

if [[an]] the ambiguity creating location is encountered while backtracking, either ignoring the execution event or bridging the ambiguity creating location.

4. (Cancelled)

5. (Currently Amended) The method of claim 1, wherein the ~~unambiguous skid region does not include an ambiguity creating location and wherein the~~ ambiguity creating location is one of:

a jump target location;

an indirect branch target location;

a branch target location;

entry point location;

a trap handler location; and

an interrupt handler location.

6. (Original) The method of claim 1, wherein the preceding operation corresponds to a load instruction; and

wherein the execution event is a cache miss.

7. (Original) The method of claim 1, wherein the preceding operation corresponds to a memory access instruction; and

wherein the execution event is either a hit or miss at a level in a memory hierarchy.

8. (Previously Presented) The method of claim 1, wherein the execution event is either an overflow or an underflow of a hardware counter.

9. (Previously Presented) The method of claim 1, wherein the execution event triggers either an overflow or an underflow of a hardware counter that is itself detected.

10. (Original) The method of claim 1, wherein the latency includes that associated with delivery of a trap.

11. (Original) The method of claim 1, wherein the latency includes that associated with delivery of a counter overflow event signal.

12. (Original) the method of claim 1, wherein the latency is associated with pipeline execution skid.

13. (Original) the method of claim 1, wherein the latency is associated with completion of in-flight operations.

14. (Original) The method of claim 1, embodied in a computer program product.

15. (Original) the method of claim 1, embodied in a least one of:

a profiling tool;

a code optimizer; and

a runtime library.

16. (Original) The method of claim 1, employed in combination with a compiler that pads the code with one or more padding operations to absorb at least some instances of the latency.

17. (Original) The method of claim 16, wherein the padding operations are not themselves associated with the execution event.

18. (Currently Amended) The method of claim 16, wherein each padding operation is not [[an]] the ambiguity creating location.

19. (Currently Amended) A method of identifying operations associated with execution events of a processor, the method comprising:

identifying a target operation and an ambiguity creating location within an execution sequence of operations of the processor;

from a point in [[an]] the execution sequence of the operations of the processor, the point coinciding with an execution event, backtracking through the operations toward a particular operation that precedes the coinciding point by a displacement ~~within an unambiguous skid region~~, wherein the displacement is based, at least in part on a type of operation appropriate for triggering the execution event; [[and]]

determining that the ambiguity creating location is not encountered while backtracking; and

associating the execution event with the particular operation.

20. (Previously Presented) The method of claim 19, further comprising:

executing the sequence of operations on the processor; and

detecting the execution event.

21. (Previously Presented) The method of claim 19, wherein the operations are instructions executable on the processor; and

wherein the particular operation is a particular one of the instructions that triggers the execution event.

22. (Previously Presented) The method of claim 19, wherein the operations correspond to instructions of program code.

23. (Previously Presented) The method of claim 19, wherein the execution event is an exception triggering execution of the particular operation.

24. (Original) The method of claim 19, wherein the execution event is a cache miss.

25. (Previously Presented) The method of claim 19, wherein the displacement is based, at least in part, on a trap delivery delay.

26. (Previously Presented) The method of claim 19, wherein the execution event triggers a hardware event and the displacement is based, at least in part, on delivery of a signal associated therewith.

27. (Original) The method of claim 26, wherein the hardware event is either underflow or overflow of a counter associated with the execution event.

28. (Original) The method of claim 19, wherein the execution event is either underflow or overflow of a counter.

29. (Previously Presented) The method of claim 19, wherein one or more instances of an intervening target of a control transfer is identified in the execution sequence of operations to facilitate the backtracking.

30. (Original) The method of claim 29, wherein at least some of the instances of intervening control transfer targets are resolved using branch history information.

31. (Currently Amended) The method of claim 19, wherein [[an]] the ambiguity creating location in the execution sequence is identified by a compiler.

32-36. (Cancelled)

37. (Previously Presented) The method of claim 19, wherein the particular operation comprises a memory referencing instruction.

38. (Previously Presented) The method of claim 37, wherein the memory referencing instruction comprises one or more of loads, stores and prefetches.

39-45. (Cancelled)

46. (Currently Amended) A method of preparing code for a processor, the method comprising:

preparing a tangible first executable instance of the code, the preparing identifying at least ambiguity creating locations therein;

executing the first executable instance and responsive to detection of an execution event, backtracking a displacement through the code to an operation thereof, wherein the

displacement is based, at least in part, on a type of operation appropriate to have triggered the execution event;

associating the operation with the execution event; and

classifying the execution event as ~~ambiguous or unambiguous~~ associated with one or more of the ambiguity creating locations or not associated with one or more of the ambiguity creating locations.

47. (Previously Presented) The method of claim 46, wherein the association between the associated operation and the execution event is based on a set of additional detections and responsive backtracking.

48. (Cancelled)

49. (Original) The method of claim 46, further comprising:

resolving at least some intervening ones of the identified ambiguity creating locations using branch history information.

50. (Currently Amended) A computer program product encoded in one or more computer readable media, the computer program product comprising:

an execution sequence of operations; and

one or more padding operations following a first operation of the execution sequence, the padding operations providing an unambiguous skid region of the execution sequence between the first operation and a subsequent ambiguity creating location to provide a displacement between the subsequent ambiguity creating location and the first operation[[s]] to cover an expected detection latency of the first operation.

51. (Previously Presented) The computer program of claim 50, wherein the first operation includes a memory access operation.

52. (Original) The computer program product of claim 50, wherein the padding operations include nops.

53. (Original) The computer program product of claim 50, wherein the unambiguous skid region does not include an ambiguity creating location.

54. (Previously Presented) The computer program product of claim 50, wherein the one or more computer readable media are selected from the set of a disk, tape, magnetic, optical, semiconductor or electronic storage medium.

55-58. (Cancelled)

59. (Currently Amended) An apparatus comprising:

means for identifying an ambiguity creating location;

means for backtracking a displacement ~~within an unambiguous skid region~~, from a point coinciding with an execution event in an execution sequence of operations on a processor, through the execution sequence toward a particular operation thereof that precedes the coinciding point, wherein the backtracking displacement is based, at least in part, on a type of the particular operation; [[and]]

means for determining that an ambiguity creating location is not encountered during the backtracking;

means for associating the execution event with the particular operation.

60. (Original) the apparatus of claim 59, further comprising:

means for bridging at least some ambiguity creating locations.

61. (Previously Presented) An apparatus comprising:

a code preparation facility suitable for preparation of an execution sequence of operations for a processor; and

means for padding the execution sequence to provide an unambiguous skid region between a particular operation and a subsequent ambiguity creating location within the sequence of operations to cover an expected detection latency, wherein the expected detection latency is based, at least in part, on a type of the particular operation.

62. (Previously Presented) The method of claim 3, wherein the ambiguity creating location is bridged using branch history information.

63. (Currently Amended) The method of claim 1 further comprising:

identifying the preceding operation and [[an]] the ambiguity creating location subsequent to the identified preceding operation;

determining that an unambiguous interval between the ambiguity creating location and the identified operation is insufficient to cover an expected detection latency for the identified preceding operation; and

inserting padding operations, which provide at least a portion of a skid region between the identified preceding operation and the ambiguity creating location, into the code subsequent to the identified operation, wherein the expected detection latency is based, at least in part, on a type of the identified preceding operation.

64. (Currently Amended) The method of claim 19 ~~further comprising identifying target operations and ambiguity creating locations in the sequence of operations~~, wherein the identified target operation[[s]] includes the particular operation.

65. (Currently Amended) The method of claim 64 further comprising ignoring a second execution event if [[an]] the ambiguity creating location is encountered while backtracking.

66. (Currently Amended) The method of claim 64 further comprising:

encountering [[an]] the ambiguity creating location while backtracking; and

bridging the ambiguity creating location using branch history information.

67. (Cancelled)

68. (Currently Amended) The method of claim [[67]] 64 further comprising inserting one or more padding operations into the sequence of operations between ~~a first of~~ the identified target operation[[s]] and ~~a first of~~ the ~~identified~~ ambiguity creating location[[s]] to cover an expected detection latency of the ~~first identified~~ target operation.

69. (Previously Presented) The method of claim 46, wherein the preparing comprises:

inserting one or more padding operations between the operation and a first of the identified ambiguity creating locations that is subsequent to the operation, wherein the one or more padding operations provide at least a portion of a skid region between the operation and the first identified ambiguity creating location sufficient to cover an expected detection latency of the operation.